**Questions**

**1) Cosa si intende per database?**

Un **database** (o base dati), è un duplice strumento:

• un oggetto fisico con memoria (un contenitore dove si salvano informazioni).

• uno strumento che permette di organizzare, strutturare, interrogare e manipolare dati, secondo uno schema logico, coerente e garantisce persistenza.

Un DB può essere relazionale o non relazionale; **SQL** è un database **relazionale** (perché le tabelle che lo compongono sono collegate tra loro). Le caratteristiche di un DB sono la **facilità** di **accesso**, **gestione** e **aggiornamento**. Deve evitare la **ridondanza** e l'**inconsistenza** dei dati, e garantire **integrità** e **sicurezza**.

Un DB è strutturato in tabelle (**entità**) che contengono colonne (**attributi**/**campi**) e righe (**record**), cioè set informativi della tabella stessa. Dopo aver creato la struttura, i dati vengono memorizzati per essere recuperati e utilizzati in operazioni (**transazioni**).

Le transazioni dei dati avvengono tramite operazioni **CRUD**:

• **Create**: con CREATE TABLE si crea la tabella e con INSERT INTO si inseriscono i dati (es. registrazione cliente).

• **Read**: con SELECT e altri comandi (WHERE, GROUP BY, ORDER BY) si leggono e interrogano i dati (es. ordini di vendita del mese corrente).

• **Update**: con UPDATE si aggiornano i dati (es. correzione di un ordine).

• **Delete**: con DELETE si eliminano i dati (es. eliminazione di un ordine cliente).

Chi progetta il DB deve garantire il rispetto del concetto **ACID**:

▪ **Atomicità**: le transazioni sono eseguite come unità singole (tutte o nessuna operazione applicata).

▪ **Coerenza**: le transazioni lasciano il DB in uno stato coerente, rispettando le regole definite.

▪ **Isolamento**: le transazioni non interferiscono tra loro, sono gestite sequenzialmente.

▪ **Durabilità**: il risultato delle transazioni è persistente (modifiche permanenti dopo il commit).

**2) Cos’è un DBMS?**

Il DB (inteso come modello) è realizzato su un sistema, che è detto DBMS (DataBase Management System):

è un software, che è progettato per farci creare, manipolare e interrogare il DB. Quindi, per utilizzare DB a livello fisico, si utilizza un software (DBMS).

Quando parliamo di DB relazionali (come in SQL), sarà RDBMS (Relational DataBase Management System).

**3) Indica le principali clausole di uno statement SELECT in ordine di esecuzione logica. Descrivi per ciascuna delle clausole indicate la logica di funzionamento.**

Le clausole (comandi o parole riservate) scritte in una query SQL, seguono un ordine sintattico preciso, ma vengono eseguite in un ordine logico.

**Ordine sintattico:**

**SELECT** …, **FROM** …, **JOIN** …, **WHERE** …, **GROUP** **BY** …, **HAVING** …, **ORDER** **BY** …;

**Ordine logico:**

**1- FROM:** Specifica la tabella (o le tabelle tramite JOIN) da cui vengono recuperati i dati. È il primo passaggio in cui il motore del DB identifica quali record interrogare.

**2- JOIN:** Combina righe di due o più tabelle, basate su una condizione (ON) correlata tra di esse. La combinazione delle tabelle avviene dopo il passaggio FROM.

**3- WHERE**: Filtra le righe restituite dalle clausole sopra, basate su una condizione specificata (lavora sui dati non aggregati). Questo passaggio viene eseguito dopo la combinazione delle tabelle e riduce il numero di righe da considerare nelle fasi successive.

**4- GROUP BY:** Raggruppa le righe con valori identici, in colonne specificate. Utilizzato con funzioni di aggregazione (COUNT, SUM, AVG) per produrre una combinazione univoca per ciascun gruppo dei campi indicati e viene chiamato sui campi che rimangono fuori dall’aggregazione.

**5- HAVING:** Filtra i gruppi di dati creati dalla clausola GROUP BY, basati su una condizione (lavora su dati aggregati). Questa clausola è simile a WHERE, ma si applica ai gruppi piuttosto che alle singole righe.

**6- SELECT**: Specifica i campi da includere nel result set (output della query). Sebbene sia la prima clausola in ordine sintattico, è eseguita dopo tutte le clausole precedenti in modo che i dati finali siano correttamente filtrati e raggruppati.

**7- ORDER BY**: Ordina i risultati in base a una o più colonne specificate. Questo passaggio viene eseguito alla fine per organizzare i dati in modo crescente o decrescente.

**4) Descrivi, immaginando uno scenario a te familiare, il concetto di GROUP BY. Utilizza l’approccio che ritieni più efficiente per trasmettere il concetto (suggerimento: disegna anche una sola tabella in Excel o in word con poche colonne e pochi record e descrivi, basandosi sulla tabella stessa, un esempio di GROUP BY).**

**GROUP BY** è una clausola utilizzata per raggruppare righe in base a una o più colonne e applicare funzioni di aggregazione (COUNT, SUM, AVG, MIN, MAX).

Esempio: Abbiamo una tabella “Sales”, che registra le vendite di un negozio di giocattoli

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Nome** | **Categoria** | **Quantità** | **Prezzo** | **ClienteID** |
| 1 | Lego | Costruzioni | 10 | 30 | 1 |
| 2 | Barbie | Bambole | 5 | 20 | 2 |
| 3 | Puzzle | Puzzle | 8 | 15 | 3 |
| 4 | Action Figure | Giocattoli d'azione | 7 | 25 | 4 |
| 5 | Play-Doh | Creatività | 6 | 10 | 2 |
| 6 | Puzzle | Puzzle | 12 | 15 | 3 |

Per calcolare il numero totale di giocattoli venduti per ciascuna categoria, usiamo GROUP BY sulla colonna "Categoria" e la funzione SUM sulla colonna "Quantità". La query SQL sarà:

**SELECT** Categoria, SUM(Quantità) **AS** Totale\_Giocattoli\_Venduti **FROM** Sales **GROUP** **BY** Categoria;

Il risultato della query sarà una nuova tabella che mostra il totale di giocattoli venduti per ogni categoria:

|  |  |
| --- | --- |
| **Categoria** | **Totale\_Giocattoli\_Venduti** |
| Costruzioni | 10 |
| Bambole | 5 |
| Puzzle | 20 |
| Giocattoli d'azione | 7 |
| Creatività | 6 |

Le righe della tabella "Sales" sono raggruppate per "Categoria", e la funzione SUM calcola la somma delle quantità di giocattoli venduti per ciascuna categoria. Notiamo ad esempio, che è stato venduto un totale di 20 giocattoli della categoria "Puzzle".

**5) Descrivi la differenza tra uno schema OLTP e uno schema OLAP.**

**Schema OLTP (OnLine Transaction Processing)**:

• Gestisce dati transazionali (Data Entry).

• Elevata velocità di lettura e scrittura per molte transazioni contemporanee.

• Garantisce consistenza, integrità e sicurezza delle transazioni.

• Tabelle altamente normalizzate per ridurre la ridondanza.

• Operazioni principali: INSERT, UPDATE, DELETE, SELECT.

**Schema OLAP (OnLine Analytical Processing)**:

• Analisi complessa di grandi volumi di dati (gestione dati analitica).

• Ottimizzato per query di lettura di grandi volumi di dati, meno frequenti.

• Utilizza Data Warehouse per l'analisi e la reportistica.

• Popolato con procedure ETL (Extract, Transform, Load).

• Operazioni principali: complesse query di SELECT per aggregare, filtrare, raggruppare e ordinare dati.

**Differenza**: sono due tipi di architetture di DB progettate per scopi differenti: OLTP è progettato per operazioni transazionali quotidiane e rapide, mentre OLAP è per analisi e reportistica di grandi volumi di dati storici.

**6) Dato un medesimo scenario di analisi, qual è la differenza in termini di risultato ottenibile tra una JOIN e una SUBQUERY?**

Prendiamo in considerazione la tabella precedente “Sales” e la tabella sotto “Customer”:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Nome** | **Cognome** | **Località** |
| 1 | Giovanni | Rossi | Roma |
| 2 | Laura | Bianchi | Milano |
| 3 | Marco | Verdi | Napoli |
| 4 | Lucia | Neri | Torino |

• **JOIN** (Utilizzata per combinare righe da due o più tabelle, basate su una condizione correlata 🡪 ON)

**Esempio**: Abbiamo una tabella "Customer" con informazioni sui clienti che hanno acquistato giocattoli.

Query SQL con INNER JOIN:

**SELECT** s.Nome, s.Categoria, s.Quantità, s.Prezzo, c.Nome, c.Cognome, c.Località **FROM** Sales s

**INNER** **JOIN** Customer c **ON** s.ClienteID = c.ID;

**Risultato**: il nome del giocattolo, la categoria, la quantità, il prezzo, il nome del cliente e la località del cliente.

Utile per visualizzare informazioni da entrambe le tabelle, insieme in un'unica query.

• **SUBQUERY** (Una query annidata all'interno di un'altra query, usata per calcolare un valore da utilizzare nella

query principale)

**Esempio**: Calcolare il prezzo medio di vendita per ciascuna categoria e visualizzarlo insieme ai dettagli vendite.

Query SQL con SUBQUERY:

**SELECT** Nome, Categoria, Quantità, Prezzo,

(**SELECT** AVG(Prezzo) **FROM** Sales s2 **WHERE** s2.Categoria = s1.Categoria) **AS** PrezzoMedioCategoria

**FROM** Sales s1;

**Risultato**: il nome del giocattolo, la categoria, la quantità, il prezzo e il prezzo medio della categoria.

Utile per calcolare e visualizzare valori intermedi o aggregati specifici per ciascun record.

**Differenze chiave**:

▪ **JOIN**: Combina dati da più tabelle e restituisce un risultato con colonne da tutte le tabelle coinvolte.

▪ **SUBQUERY**: Calcola valori intermedi o aggregati da utilizzare nella query principale, ideale per calcoli specifici per ogni record.

**7) Cosa si intende per DML e DDL?**

**DML** e **DDL** si riferiscono a due tipi distinti di linguaggi utilizzati per interagire con i database.

**DML (Data Manipulation Language)**:

• Sottoinsieme del linguaggio SQL, utilizzato per manipolare i dati (contenuto).

• Interroga o modifica i dati.

• Operazioni principali:

* INSERT: Aggiunge nuovi record.
* UPDATE: Modifica dati esistenti.
* DELETE: Elimina record.
* SELECT: Recupera dati.

**DDL (Data Definition Language)**:

• Sottoinsieme del linguaggio SQL, utilizzato per definire e gestire la struttura del DB.

• Operazioni principali:

* CREATE: Crea nuovi oggetti (tabelle, indici).
* ALTER: Modifica struttura di oggetti esistenti.
* DROP: Elimina oggetti esistenti.
* TRUNCATE: Rimuove tutti i record, ma mantiene la struttura della tabella.

**8) Quali istruzioni possono utilizzare per estrarre l’anno da un campo data? Proponi degli esempi.**

Con la funzione **YEAR()** si può estrarre l'anno da un campo data.

Esempio per estrarre anno corrente: **SELECT** **YEAR**(CURDATE()) **AS** AnnoCorrente;

Esempio: Supponiamo di avere la seguente tabella “Sales”

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Nome** | **Categoria** | **Quantità** | **Prezzo** | **DataVendita** | **ClienteID** |
| 1 | Lego | Costruzioni | 10 | 30 | 15/03/2024 | 1 |
| 2 | Barbie | Bambole | 5 | 20 | 10/05/2024 | 2 |
| 3 | Puzzle | Puzzle | 8 | 15 | 22/11/2023 | 3 |
| 4 | Action Figure | Giocattoli d'azione | 7 | 25 | 19/07/2023 | 4 |
| 5 | Play-Doh | Creatività | 6 | 10 | 25/01/2024 | 2 |
| 6 | Puzzle | Puzzle | 12 | 15 | 24/01/2025 | 3 |

Ecco come estrarre l'anno dalla colonna **DataVendita**:

**SELECT** Nome, Categoria, Quantità, Prezzo, **YEAR**(DataVendita) **AS** AnnoVendita **FROM** Sales;

**9) Qual è la differenza tra gli operatori logici AND e OR?**

Gli operatori logici **AND** e **OR** sono utilizzati per combinare più condizioni, in una clausola WHERE.

• **Operatore AND**: restituisce le righe che soddisfano tutte le condizioni specificate. Quindi, tutte le condizioni

combinate con **AND** devono essere vere, affinché una riga venga inclusa nei risultati.

Esempio:

**SELECT** \* **FROM** Sales **WHERE** Categoria = 'Puzzle' **AND** Prezzo < 20;

Questa query restituirà solo i record della tabella **Sales**, dove la categoria è 'Puzzle' **e** il prezzo è inferiore a 20.

• **Operatore OR**: restituisce le righe che soddisfano almeno una delle condizioni specificate. Quindi, se almeno una

delle condizioni combinate con **OR** è vera, la riga verrà inclusa nei risultati.

Esempio:

**SELECT** \* **FROM** Sales **WHERE** Categoria = 'Puzzle' **OR** Prezzo < 20;

Questa query restituirà i record della tabella **Sales**, dove la categoria è 'Puzzle' **o** il prezzo è inferiore a 20.

Le righe saranno incluse se soddisfano almeno una delle due condizioni.

**N.B.** Si posso anche combinare gli operatori **AND** e **OR** insieme, per creare query più complesse. In questi casi, si utilizzano le parentesi per chiarire l'ordine delle operazioni.

**10) È possibile innestare una query nella clausola SELECT?**

Una "subquery" o "inner query" è utilizzata per calcolare un valore da usare nella query principale.

**Esempio di Subquery nella Clausola SELECT con la tabella Sales:**

**SELECT** Nome, Categoria, Prezzo,

(**SELECT** SUM(Quantità) **FROM** Sales **AS** s2

**WHERE** s2.Nome = s1.Nome) **AS** TotaleVendite **FROM** Sales **AS** s1;

• La subquery (**SELECT** SUM(Quantità) **FROM** Sales **AS** s2 **WHERE** s2.Nome = s1.Nome) calcola la somma totale

delle quantità vendute per ciascun giocattolo.

• La subquery viene eseguita per ogni riga della tabella Sales principale (AS s1), utilizzando il nome del giocattolo

come condizione di corrispondenza.

**Risultato**: il nome del giocattolo, la categoria, il prezzo e il totale delle vendite per ciascun giocattolo.

**11) Qual è la differenza tra l’operatore logico OR e l’operatore logico IN?**

• **Operatore OR**: Combina più condizioni nella clausola WHERE e restituisce righe che soddisfano almeno una

condizione.

**Esempio**:

**SELECT** \* **FROM** Sales **WHERE** Categoria = 'Puzzle' **OR** Prezzo < 20;

• **Operatore IN**: Verifica se un valore è all'interno di un elenco specificato di valori, rendendo i confronti multipli più semplici e leggibili sintatticamente.

**Esempio**:

**SELECT** \* **FROM** Sales **WHERE** Categoria IN ('Puzzle', 'Costruzioni');

**12) L’operatore logico BETWEEN include anche gli estremi del range specificato?**

Sì, l'operatore logico **BETWEEN** include anche gli estremi del range specificato. Quando utilizzi BETWEEN in una query SQL, i valori iniziale e finale del range sono inclusi nei risultati.

Esempio:

**SELECT** \* **FROM** Sales

**WHERE** Prezzo **BETWEEN** 10 **AND** 20;

Questa query restituirà i record della tabella Sales dove il prezzo è compreso tra 10 e 20, inclusi 10 e 20.

**DESCRIZIONE DATABASE**

Ho creato database che si chiama ToysGroup, per la gestione ed analisi vendite di prodotti di un’azienda che distribuisce articoli (giocatoli) in diverse aree geografiche del mondo.

**Tabelle e relativi Attributi:**

• **Category:** contiene 10 record;

* **Attributi**: • category\_id(PK); • category\_name; • theme; • functionality.

• **Product:** contiene 25 record;

* **Attributi**: • product\_id(PK); • product\_name; • age\_range; • unit\_quantity; • unit\_price; • standard\_cost;

• category\_id(FK) -> category(category\_id).

• **Region:** contiene 20 record;

* **Attributi**: • region\_id(PK); • region\_name.

• **State:** contiene 22 record;

* **Attributi**: • state\_id(PK); • state\_name; • city;• region\_id(FK) -> region(region\_id).

• **Customer:** contiene 20 record;

* **Attributi**: • customer\_id (PK); • first\_name; • last\_name; • address; • postal\_code • email;

• phone\_number; • state\_id (FK) -> state(state\_id).

• **Sales:** contiene 22 record;

* **Attributi**: • sale\_id (PK); • order\_number; • order\_date; • sale\_date; • quantity\_sold; • unit\_price;

• total\_amount; • product\_id (FK) -> product(product\_id);

• customer\_id (FK) -> customer(customer\_id); • region\_id (FK) -> region(region\_id).

**Relazioni e Cardinalità:**

• **Category – Product**

**Relazione**: Uno a Molti (1:N)

**Cardinalità**: Category (0,1) -> Product (0,N)

**Commento**: Una categoria può includere uno o più prodotti, ma un prodotto appartiene a una sola categoria o a

nessuna.

• **Product – Sales**

**Relazione**: Uno a Molti (1:N)

**Cardinalità**: Product (0,N) -> Sales (1,N)

**Commento**: Un prodotto può essere venduto tante volte (o nessuna) ed è contenuto in una o più transazioni di

vendita. Ogni transazione di vendita è riferita a un solo prodotto.

• **Customer – Sales**

**Relazione**: Uno a Molti (1:N)

**Cardinalità**: Customer (0,1) -> Sales (0,N)

**Commento**: Un cliente può effettuare molte transazioni di vendita, ma ciascuna transazione di vendita è riferita

a un solo cliente.

• **Region – Sales**

**Relazione**: Uno a Molti (1:N)

**Cardinalità**: Region (0,1) -> Sales (0,N)

**Commento**: Possono esserci molte o nessuna transazione per ciascuna regione. Ciascuna transazione di vendita

è riferita a una sola regione.

• **Region – State**

**Relazione**: Uno a Molti (1:N)

**Cardinalità**: Region (0,1) -> State (0,N)

**Commento**: Una regione può includere uno o più stati, ma ciascuno stato appartiene a una sola regione.

• **State – Customer**

Relazione: Uno a Molti (1:N)

Cardinalità: State (0,1) -> Customer (0,N)

**Commento**: Uno stato può includere uno o più clienti, ma ciascun cliente risiede in uno solo stato.

**MODELLO E/R**

Immagine che contiene testo, schermata, Carattere, Parallelo

Il contenuto generato dall'IA potrebbe non essere corretto.